DISQUO: A Distributed 100% Throughput Algorithm for a Buffered Crossbar Switch

†Shunyuan Ye, ‡Yanming Shen, †Shivendra Panwar
†Department of Electrical and Computer Engineering, Polytechnic Institute of NYU
‡School of Computer Science and Technology, Dalian University of Technology, China
e-mail: sye02@students.poly.edu, syanming@photon.poly.edu, panwar@catt.poly.edu

***Abstract*—The promise of a buffered crossbar switch - a crossbar switch with a packet buffer at each crosspoint - is that it can provide good delay performance with much less complex, practical scheduling algorithms. With today's technology, it is now possible to implement it in a single chip. Thus it has attracted great attention recently. Though simple distributed algorithms can achieve 100% throughput under uniform traffic, so far there are no distributed algorithms which can achieve 100% throughput under general admissible arrival patterns. In this paper, we propose a distributed scheduling algorithm which achieves 100% throughput for any admissible Bernoulli arrival traffic. To the best of our knowledge, this is the first distributed algorithm which can achieve this. The algorithm is called DISQUO: DIStributed QUeue input-Output scheduler. Our simulation results also show that DISQUO can provide good delay performance for different traffic patterns.**

I. Introduction

The fast growing traffic demand in the Internet requires that packet switches should be simple, fast and efficient. Due to the memory speed limit, most current switches use *input queuing* (IQ) or *combined input and output queuing* (CIOQ), with a bufferless crossbar switching fabric. The scheduler must find a matching between inputs and outputs. Such switches require centralized, sometimes complex, algorithms to achieve good performance, such as *maximal* [1] and *maximum weight matching* [2]. Maximum weight matching can achieve 100% throughput for any admissible arrival traffic, but it is not practical to implement due to its high complexity. Maximal matching, on the other hand, cannot achieve as high a throughput as maximum weight matching. A number of practical iterative algorithms have been proposed, such as iSLIP [3] and DRRM [4]. iSLIP uses multiple iterations to converge to a maximal matching. DRRM can achieve 100% throughput only under i.i.d. and uniform traffic. EMHW [5] has been proved to stabilize the system for any admissible traffic, but it is still centralized and has a complexity of $O(logN)$.

With today's ASIC technology, it is now possible to add small buffers at each crosspoint inside the crossbar. This makes the *buffered crossbar* or *combined input and crossbar queueing* (CICQ) switch a much more attractive architecture since its scheduler is potentially much simpler. Each input(output) knows the state of all crosspoint buffers to(from) which it can send(receive) packets. The input and output schedulers can be independent. First, each input picks a crosspoint buffer to send a packet to. Then, each output picks a crosspoint buffer to transmit a packet from, as shown in Fig. 1. A centralized scheduler is not needed since the processing can be distributed at each input and output. It has been shown that simple algorithms such as *round robin* at both the inputs and outputs (RR-RR) [6], or *longest queue first* at the inputs, and *round robin* at the outputs (LQF-RR) [7], can provide 100% throughput under *uniform* traffic. SQUISH and SQUID [8] can achieve 100% throughput for any admissible traffic, but these are centralized algorithms which do not scale with the increase in the number of ports due to the communication complexity and latency. Thus, it is impossible to implement these algorithms in large scale high-speed switching systems.

Recently, it has been shown that CSMA-like algorithms [9], [10] can achieve the maximum throughput in wireless ad hoc networks in continuous time systems. These ideas were extended to discrete time systems by Ni and Srikant in [11]. Inspired by the CSMA-like algorithms, in this paper, we propose a distributed algorithm in buffered crossbar switches that can stabilize the system under any admissible Bernoulli traffic matrix. The scheduling algorithm is called DISQUO: DIStributed QUeue input-Output scheduler. With DISQUO, no message passing is required. Each input only uses its local queue information and the previous time slot schedule to make its scheduling decision. We prove the stability of the system and evaluate the performance of DISQUO by running extensive simulations. To the best of our knowledge, this is the first distributed algorithm which can achieve 100% throughput under any admissible Bernoulli traffic in buffered crossbar switches. The simulation results also show that it can provide good delay performance as compared to output-queued switches, under different types of traffic.

The rest of paper is organized as follows. We first briefly describe the buffered crossbar switch in Sec. II. DISQUO is presented in Sec. III, and an example is provided to illustrate its operation. We derive the stationary distribution of the system in Sec. IV, and prove its stability in Sec. V. Simulations results are presented in Sec. VI to show its delay performance.

II. Crosspoint Buffered Switch

An $N \times N$ switch is shown in Fig. 1. We assume fixed size packet(cell) switching. Variable size packets can be segmented into cells before switching and reassembled at the output ports.

This work is supported by the National Science Foundation and the New York State Center for Advanced Technology in Telecommunications (CATT).

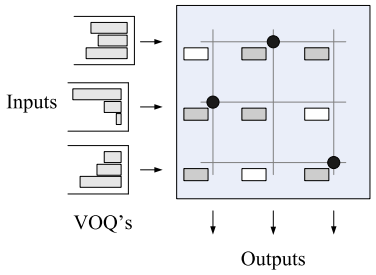

Fig. 1: Buffered Crossbar Switch

There are *virtual output queues (VOQs)* at the inputs to prevent head-of-line blocking. Each input maintains N VOQs, one for each output. Let VOQ_{ij} represent the VOQ at input i for output j. Let $Q_{ij}(n)$ denote the queue length of VOQ_{ij} at time n. Let (i, j) represent the crosspoint between input i and output j. Note that a VOQ corresponds to a crosspoint.

Each crosspoint has a buffer of size K. But $K = 1$ is sufficient for our algorithm, and most current implementations are constrained in the size of K. We will therefore assume that $K = 1$ in the following. Let CB_{ij} denote the buffer of the crosspoint between input i and output j. $B_{ij}(n) \in \{0, 1\}$ denotes the occupancy of CB_{ij} at time n.

A schedule can be represented by $\mathbf{S}(n) = [\mathbf{S}^I(n), \mathbf{S}^O(n)]$. $\mathbf{S}^I(n) = [S^I_{ij}(n)]$ is the input schedule. Each input port can only transmit at most one cell at each time slot. Thus the input schedule is subject to the following constraints:

$$\sum_j S^I_{ij}(n) \le 1,\ S^I_{ij}(n) = 0 \text{ if } B_{ij}(n) = 1. \tag{1}$$

$\mathbf{S}^O(n) = [S^O_{ij}(n)]$ is the output schedule. It has to satisfy the following constraints:

$$\sum_i S^O_{ij}(n) \le 1,\ S^O_{ij}(n) = 0 \text{ if } B_{ij}(n) = 0. \tag{2}$$

Let λ_{ij} represent the arrival rate of traffic between input i and output j. We assume that the arrival process is Bernoulli.

Definition 1: An arrival process is said to be *admissible* if it satisfies:

$$\sum_j \lambda_{ij} < 1, \text{ and } \sum_i \lambda_{ij} < 1. \tag{3}$$

III. The DISQUO Scheduling Algorithm

In this section, we first define the notation used in our algorithm. We then present the DISQUO scheduling algorithm. To help present the idea, we first describe the basic concept of DISQUO and then show how it can be implemented in a distributed manner by utilizing the state of the crosspoint buffers. An example is then given to illustrate how it works.

A. Notation

Definition 2: A *DISQUO schedule* $\mathbf{X}(n)$ is an $N \times N$ matrix, where $X_{ij}(n) \in \{0, 1\}$, and $\sum_i X_{ij}(n) \le 1$, $\sum_j X_{ij}(n) \le 1$.

With some abuse of notation, we also use $\mathbf{X}$ to represent a set, and write $(i, j) \in \mathbf{X}$ if $X_{ij} = 1$. Note that a DISQUO schedule $\mathbf{X}$ has the property that if $X_{ij} = 1$, then $\forall i' \ne i$, $X_{i'j} = 0$ and $\forall j' \ne j$, $X_{ij'} = 0$. We define these crosspoints as its *neighbors*.

Definition 3: For a crosspoint (i, j), its neighbors are defined as:

$$\mathcal{N}(i, j) = \{(i', j) \text{ or } (i, j') \mid \forall i' \ne i,\ \forall j' \ne j\} \tag{4}$$

A DISQUO schedule $\mathbf{X}$ then has the following property:

Property 1: If $(i, j) \in \mathbf{X}$, $\forall (k, l) \in \mathcal{N}(i, j)$, $(k, l) \notin \mathbf{X}$.

We also define the DISQUO schedule to have the following properties:

Property 2: At each time slot, when a DISQUO schedule is generated, each input and output port determine their schedules by observing the following rules:

- For input i, when $X_{ij}(n) = 1$, if $Q_{ij}(n) > 0$ and $B_{ij}(n-1) = 0$, then $S^I_{ij}(n) = 1$. Otherwise, $S^I_{ij}(n) = 0$.
- For output j, if $X_{ij}(n) = 1$ and $B_{ij}(n) > 0$, $S^O_{ij}(n) = 1$.

Property 3: For an input i, if $\forall j$, $X_{ij} = 0$, then it is referred to as a *free input*. A free input port can randomly pick an eligible crosspoint to serve, i.e. it can transfer a packet to any free crosspoint buffer.

Property 4: For an output port j, if $\forall i$, $X_{ij} = 0$, then it is a *free output*. A free output can randomly pick a non-empty crosspoint to serve.

Let $\mathcal{X}$ represent the set of all DISQUO schedules.

B. The Basic DISQUO Algorithm

The initial DISQUO schedule $\mathbf{X}(0)$ can be any schedule that satisfies Definition 2. For a switch of size N, there are $N!$ distinct matchings. A Hamiltonian walk schedule $\mathbf{H}(n)$ visits each of the $N!$ distinct matchings exactly once during $N!$ slots. A distributed Hamiltonian walk can be simply generated with a time complexity of $O(1)$ [12]. Note that $\mathbf{H}(n)$ is also a DISQUO schedule.

The DISQUO schedule $\mathbf{X}(n)$ then is generated by merging $\mathbf{X}(n-1)$ and $\mathbf{H}(n)$ as follows:

Basic DISQUO Scheduling Algorithm

- $\forall\ (i, j) \notin \mathbf{H}(n)$:
  (a) $X_{ij}(n) = X_{ij}(n-1)$.
- For $(i, j) \in \mathbf{H}(n)$:
  - If $(i, j) \in \mathbf{X}(n-1)$:
    (b) $X_{ij}(n) = 1$ with probability p_{ij};
    (c) $X_{ij}(n) = 0$ with probability $\overline{p}_{ij} = 1 - p_{ij}$.
  - If $(i, j) \notin \mathbf{X}(n-1)$, and $\forall (k, l) \in \mathcal{N}(i, j)$, $X_{kl}(n-1) = 0$, then:
    (d) $X_{ij}(n) = 1$ with probability p_{ij};
    (e) $X_{ij}(n) = 0$ with probability $\overline{p}_{ij} = 1 - p_{ij}$.
  - Else, if $(i, j) \notin \mathbf{X}(n-1)$, and $\exists (k, l) \in \mathcal{N}(i, j)$ such that $X_{kl}(n-1) = 1$:
    (f) $X_{ij}(n) = 0$.

p_{ij} is a concave function (to be specified later) of the queue size Q_{ij} such that when $Q_{ij} = 0$, $p_{ij} = 0$. Note that in our algorithm, $X_{ij}(n)$ can change only when the VOQ_{ij} is selected by $\mathbf{H}(n)$.

C. Distributed Implementation

In the algorithm, each input i only needs to keep track of the DISQUO schedule in the previous slot, i.e. for which output j was $X_{ij}(n-1) = 1$. Similarly, each output only needs to keep track of for which input i was $X_{ij}(n-1) = 1$. Since the algorithm is distributed, there is no message passing between inputs and outputs. DISQUO has to make sure that if $X_{ij}(n) = 1$, both input i and output j are aware of this. Then the inputs and outputs can keep a consistent view of the DISQUO schedule. The DISQUO algorithm works as follows.

Input Scheduling Decisions

At each input port i, assume (i, j) is selected by $\mathbf{H}(\mathbf{n})$.
∘ If there exists a j', with $X_{ij'}(n-1) = 1$:
- If $j = j'$, $(i, j) \in \mathbf{X}(n-1)$ and $(i, j) \in \mathbf{H}(n)$:
  (a) $X_{ij}(n) = 1$ with probability p_{ij};
  (b) $X_{ij}(n) = 0$ with probability $\overline{p}_{ij} = 1 - p_{ij}$.
- Else,
  (c) $X_{ij}(n) = 0$.

∘ Else, if there is no j' such that $X_{ij'}(n-1) = 1$, then input i is a free input:
- If $\forall (k, l) \in \mathcal{N}(i, j)$, $X_{kl}(n-1) = 0$ (We will explain later how an input port can learn this):
  (d) $X_{ij}(n) = 1$ with probability p_{ij};
  (e) $X_{ij}(n) = 0$ with probability $\overline{p}_{ij} = 1 - p_{ij}$.
- Else,
  (f) $X_{ij}(n) = 0$.

Output Scheduling Decisions

Each output port j has to learn the scheduling decision made by the input. Assume (i, j) is selected by $\mathbf{H}(n)$.
∘ If there exists an i', with $X_{i'j}(n-1) = 1$:
- If $i = i'$, $(i, j) \in \mathbf{X}(n-1)$ and $(i, j) \in \mathbf{H}(n)$. As shown above, input i may change X_{ij} from 1 to 0. Therefore, output j has to observe the crosspoint buffer to learn the input's decision.
  (a) If input i transmits a packet to CB_{ij} at the beginning of time n, $X_{ij}(n) = 1$
  (b) Otherwise, $X_{ij}(n) = 0$.
- Else,
  (c) $X_{ij}(n) = X_{ij}(n-1) = 0$

∘ Else, if there is no i' such that $X_{i'j}(n-1) = 1$, then output j is free:
- (i) If the buffer at crosspoint (i, j) is empty and input i sends a packet to CB_{ij} at the beginning of time n; (ii) or, if the buffer is not empty, output j is required by DISQUO to transmit this packet from CB_{ij} at time n, and if then input i sends a packet to CB_{ij} at the beginning of time $n+1$, output j can update its schedule of time n as:
  (d) $X_{ij}(n) = 1$.
- Else,
  (e) $X_{ij}(n) = 0$.

So in the algorithm, the inputs are making the scheduling decisions and updating the DISQUO schedule based on $\mathbf{H}(n)$. The output ports have to learn the inputs' decisions. The key point of DISQUO is that by observing crosspoint buffers, an input and an output can learn each other's decisions implicitly. As stated in the algorithm, if a free input decides to set $X_{ij}(n) = 1$ from $X_{ij}(n-1) = 0$, it has to make sure that output j was also free so that there does not exist any $(k, l) \in \mathcal{N}(i, j)$ such that $X_{kl}(n-1) = 1$. The input can learn whether output j was free or not by observing the crosspoint buffer CB_{ij}. If it is served by output port j at time n, input i learns that the output was free at time $n-1$ and confirms that $\forall (k, l) \in \mathcal{N}(i, j)$, $X_{kl}(n-1) = 0$.

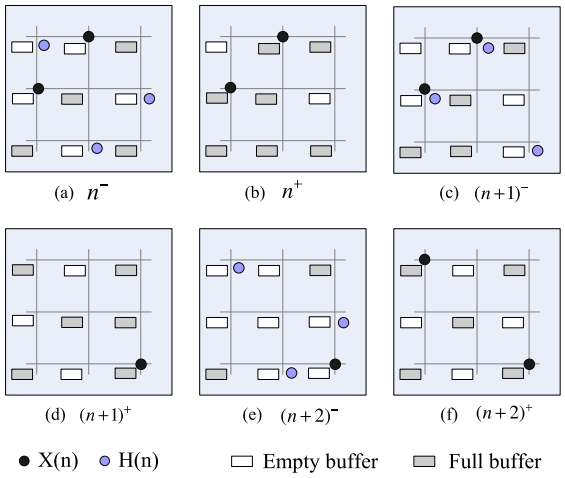

Fig. 2: An example

D. An Example

To help understand DISQUO, we give an illustrative example here. Recall that the input actions are performed at the beginning of each time slot and outputs transmit packets from the crosspoint buffers before the end of each time slot.

(1) In Fig. 2(a), the DISQUO schedule is $\mathbf{X}(n-1) = \{(1, 2), (2, 1)\}$ and the Hamiltonian walk schedule is $\mathbf{H}(n) = \{(1, 1), (2, 3), (3, 2)\}$. For input 1, $X_{12}(n-1) = 1$ and $(1, 2)$ is not selected by $\mathbf{H}(n)$, so $X_{12}(n) = X_{12}(n-1) = 1$. Similarly, $X_{21}(n) = 1$. Input 3 is free and since $(3, 2)$ is selected, it decides to send a packet to CB_{32} with probability p_{32}, and it will observe CB_{32} to see if output 2 is also free. But output 2 is not free, thus the packet in CB_{32} will not be transmitted. Input 3 can observe this by the end of time n. Thus, $X_{32}(n) = 0$. So the DISQUO schedule at time n is still $\mathbf{X}(n) = \{(1, 2), (2, 1)\}$.

(2) At time $n+1$, $\mathbf{H}(n+1) = \{(1, 2), (2, 1), (3, 3)\}$. Both $(1, 2)$ and $(2, 1)$ are selected by $\mathbf{H}(n+1)$. So input 1 and 2 change their schedules with probability $\overline{p}_{12} = 1 - p_{12}$ and $\overline{p}_{21} = 1 - p_{21}$. In the example, they both decide to change their schedules and stop sending packets to CB_{12} and CB_{21}. Therefore, $X_{12}(n+1) = 0$ and $X_{21}(n+1) = 0$. Output 2 and output 1 can learn this by observing CB_{12} and CB_{21}, respectively. Input 3 is free and $(3, 3)$ is selected by $\mathbf{H}(n+1)$. So input 3 sends a packet to CB_{33} with probability p_{33}. In this example, input 3 does send a packet to CB_{33}. Output 3

is free and it learns that $X_{33}(n+1) = 1$ by observing CB_{33}. Therefore output 3 transmits the packet from CB_{33} at time n, which is observed by input 3 and it confirms that output 3 is also free. Then the DISQUO schedule becomes $\mathbf{X}(n+1) = \{(3,3)\}$, as shown Fig. 2(d).

(3) At time $n+2$, $\mathbf{H}(n+2) = \{(1,1),(2,3),(3,2)\}$. Input 1 is free, and $(1,1)$ is selected by $\mathbf{H}(n+2)$. So it sends a packet to CB_{11} with probability p_{11}. Output 1 is also free, and it learns that $X_{11}(n+2) = 1$ by observing CB_{11}. The packet then is transmitted by output 1, and input 1 confirms that output 1 is free. So, $X_{11}(n+2) = 1$. Input 2 is free and since $(2,3)$ is selected by $\mathbf{H}(n+2)$, it has to decide whether to send a packet to CB_{23} or not with probability p_{23}. As we can see, input 2 decides not to send a packet to CB_{23}, therefore $X_{23}(n+2) = 0$. $(3,3)$ is not in $\mathbf{H}(n+2)$, so $X_{33}(n+2) = X_{33}(n+1) = 1$. The DISQUO schedule then is $\mathbf{X}(n+2) = \{(1,1),(3,3)\}$.

IV. Stationary Distribution

Lemma 1: If $\mathbf{X}(n-1) \in \mathcal{X}$, then $\mathbf{X}(n) \in \mathcal{X}$.

Proof: As defined, $\mathbf{X}$ is a DISQUO schedule if and only if $\forall(i,j)$ such that $X_{ij} = 1$: $\forall(k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin \mathbf{X}$.

For any (i,j) such that $X_{ij}(n) = 1$, it belongs to one of those two cases below:

1) $X_{ij}(n-1) = 1$.
2) $X_{ij}(n-1) = 0$ and $(i,j) \in \mathbf{H}(n)$.

If $X_{ij}(n) = X_{ij}(n-1) = 1$, $\forall(k,l) \in \mathcal{N}(i,j)$, $X_{kl}(n-1) = 0$. According to the DISQUO algorithm, whether $(k,l) \in \mathbf{H}(n)$ or not, we have $X_{kl}(n) = X_{kl}(n-1) = 0$.

If $X_{ij}(n) = 1$ and $X_{ij}(n-1) = 0$, then $(i,j) \in \mathbf{H}(n)$. According to the DISQUO algorithm, X_{ij} can change from 0 to 1 only when $\forall(k,l) \in \mathcal{N}(i,j)$, $X_{kl}(n-1) = 0$. Since $\mathbf{H}(n)$ is a DISQUO schedule, $\forall(k,l) \in \mathcal{N}(i,j)$, $(k,l) \notin \mathbf{H}(n)$. Thus, (k,l) has to keep the schedule of the previous slot unchanged. Therefore, $X_{kl}(n) = X_{kl}(n-1) = 0$.

For any (i,j) such that $X_{ij}(n) = 1$, we have proved that $\forall(k,l) \in \mathcal{N}(i,j)$, $X_{kl}(n) = 0$. So $\mathbf{X}(n) \in \mathcal{X}$ if $\mathbf{X}(n-1) \in \mathcal{X}$. **QED** ■

Lemma 2: A DISQUO schedule $\mathbf{X} \in \mathcal{X}$ can transit to a schedule $\mathbf{X}'$ in the next slot if and only if $\mathbf{X} \cup \mathbf{X}' \in \mathcal{X}$.

Proof: (Necessity): Suppose that $\mathbf{X} \cup \mathbf{X}' \notin \mathcal{X}$. According to the definition of DISQUO schedule, there exists at least one $(i,j) \in \mathbf{X} \cap \overline{\mathbf{X}'}$ and $(k,l) \in \overline{\mathbf{X}} \cap \mathbf{X}'$ such that $(k,l) \in \mathcal{N}(i,j)$. In a transition from $\mathbf{X}$ to $\mathbf{X}'$, both $(\mathbf{X} \cap \overline{\mathbf{X}'})$ and $(\overline{\mathbf{X}} \cap \mathbf{X}')$ change their states. According to the DISQUO scheduling algorithm, a VOQ can change its scheduling decision only when it is selected by $\mathbf{H}(n)$. Since both (i,j) and (k,l) change their states, they should be both in $\mathbf{H}(n)$. But $\mathbf{H}(n)$ is a DISQUO schedule generated by Hamiltonian walk, which means if $(i,j) \in \mathbf{H}(n)$ and $(k,l) \in \mathcal{N}(i,j)$, (k,l) can not be in $\mathbf{H}$(n). Hence, the initial assumption leads to a contradiction. So, a DISQUO schedule $\mathbf{X} \in \mathcal{X}$ can not transit to a schedule $\mathbf{X}'$ when $\mathbf{X} \cup \mathbf{X}' \notin \mathcal{X}$.

(Sufficiency): Suppose that $\mathbf{X}'$ is a DISQUO schedule such that $\mathbf{X} \cup \mathbf{X}' \in \mathcal{X}$. Since $(\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}') \subseteq \mathbf{X} \cup \mathbf{X}'$, we then have $(\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}') \in \mathcal{X}$. Therefore, there exists at least one $\mathbf{H}(n)$ such that $(\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}') \subseteq \mathbf{H}(n)$. When $\mathbf{X}$ is the current DISQUO schedule and the $\mathbf{H}(n)$ selected satisfies $(\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}') \subseteq \mathbf{H}(n)$, following the DISQUO algorithm, the system can make a transition to $\mathbf{X}'$ if both $(\mathbf{X} \cap \overline{\mathbf{X}'})$ and $(\overline{\mathbf{X}} \cap \mathbf{X}')$ decide to change their scheduling decisions, and other selected elements in $\mathbf{H}(n)$ decide to keep their schedules of the previous slot. This transition probability is greater than 0, which we will define in Lemma 3. **QED** ■

Lemma 3: Suppose that $\mathbf{X} \cup \mathbf{X}' \in \mathcal{X}$. Then the transition probability from $\mathbf{X}$ to $\mathbf{X}'$ is:

$$p(\mathbf{X},\mathbf{X}') = \sum_{\mathbf{H}:\mathbf{X}\triangle\mathbf{X}'\in\mathbf{H}} a(\mathbf{H}) \prod_{(i,j)\in\mathbf{X}\cap\overline{\mathbf{X}}'} \overline{p}_{ij} \prod_{(k,l)\in\overline{\mathbf{X}}\cap\mathbf{X}'} p_{kl} \bullet \prod_{(u,v)\in\mathbf{X}\cap\mathbf{X}'\cap\mathbf{H}} p_{uv} \prod_{(x,y)\in\mathbf{H}\cap\overline{\mathbf{X}\cup\mathbf{X}'}\cap\overline{\mathcal{N}(\mathbf{X}\cup\mathbf{X}')}} \overline{p}_{xy}, \quad (5)$$

where $a(\mathbf{H})$ is the probability that $\mathbf{H}$ is selected (which is $\frac{1}{N!}$), and $\mathbf{X} \triangle \mathbf{X}' = (\mathbf{X} \cap \overline{\mathbf{X}'}) \cup (\overline{\mathbf{X}} \cap \mathbf{X}')$.

Proof: Since $\mathbf{X} \cup \mathbf{X}' \in \mathcal{X}$, according to Lemma 2, the system can make a transition from $\mathbf{X}$ to $\mathbf{X}'$. The transition occurs only when the VOQs of the selected $\mathbf{H}$ satisfy the conditions below:

1) For any $(i,j) \in \mathbf{X} \cap \overline{\mathbf{X}'}$: the VOQ is selected by $\mathbf{H}$ and decides to change its scheduling decision from 1 to 0, which happens with probability $\overline{p}_{ij}$.
2) For any $(k,l) \in \overline{\mathbf{X}} \cap \mathbf{X}'$: the VOQ is selected by $\mathbf{H}$ and decides to change its scheduling decision from 0 to 1, which happens with probability p_{kl}.
3) For any $(u,v) \in \mathbf{X} \cap \mathbf{X}' \cap \mathbf{H}$: the VOQ was in the DISQUO schedule of previous time slot, and even though selected by $\mathbf{H}$ it decides to keep its schedule, which occurs with probability p_{uv}.
4) For any $(x,y) \in \mathbf{H} \cap \overline{\mathbf{X} \cup \mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X})}$: neither the VOQ nor any of its neighbors was in the DISQUO schedule of previous time slot, and though selected by $\mathbf{H}$ it decides to keep its schedule, which occurs with probability $\overline{p}_{xy}$.

Since $\mathbf{H}$ is a DISQUO schedule and $\overline{\mathbf{X}} \cap \mathbf{X}' \in \mathbf{H}$, $\mathbf{H} \cap \mathcal{N}(\overline{\mathbf{X}} \cap \mathbf{X}') = \emptyset$. Thus $\mathbf{H} \cap \overline{\mathbf{X} \cup \mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X})} = \mathbf{H} \cap \overline{\mathbf{X} \cup \mathbf{X}' \cap \mathcal{N}(\mathbf{X} \cup \mathbf{X}')}$. We replace $\mathbf{H} \cap \overline{\mathbf{X} \cup \mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X})}$ by $\mathbf{H} \cap \overline{\mathbf{X} \cup \mathbf{X}'} \cap \overline{\mathcal{N}(\mathbf{X} \cup \mathbf{X}')}$ in Eq. (5) for the proof of the stationary distribution in the following.

Since $\mathbf{H}$ is a DISQUO schedule, for any two VOQs in $\mathbf{H}$, they are not neighbors of each other. Therefore, they can make the scheduling decisions independently. We then can multiply the probabilities of all the four categories above, which leads to the transition probability given by Eq. (5). **QED** ■

As we can see from Lemma 3, the DISQUO schedule $\mathbf{X}$(n) only depends on the previous time slot $\mathbf{X}$(n-1), thus $\mathbf{X}$(n-1), $\mathbf{X}$(n), $\mathbf{X}$(n+1)$\cdots$ is a Markov chain. The state transition probability is given in Eq. (5). So we can define the Markov chain of the system based on the DISQUO schedule and will derive its stationary distribution in the following.

Lemma 4: The Markov chain of the system is positive recurrent.

Proof: Suppose that $\mathbf{X}$ is a DISQUO schedule, and it has k non-zero elements: (i_1, j_1), $(i_2, j_2) \cdots (i_k, j_k) \in \mathbf{X}$. Let $\mathbf{X}_l$ represent a DISQUO schedule which has l non-zero elements: (i_1, j_1), $(i_2, j_2) \cdots (i_l, j_l) \in \mathbf{X}_l \subseteq \mathbf{X}$, $0 \leq l \leq k$. We can see that $\mathbf{X}_0 = \mathbf{0}$ and $\mathbf{X}_k = \mathbf{X}$. Since $\mathbf{X}$ is a DISQUO schedule, $\mathbf{X}_l$ is also a DISQUO schedule and $\mathbf{X}_{l-1} \cup \mathbf{X}_l = \mathbf{X}_l \in \mathcal{X}$. Therefore, the system can make a transition from $\mathbf{X}_{l-1}$ to $\mathbf{X}_l$ with positive probability, as we already proved in Lemma 3. Hence, state $\mathbf{X}_0$ can reach any state $\mathbf{X} \in \mathcal{X}$ with positive probability in a finite number of steps and vice versa. Thus, the Markov chain is positive recurrent. **QED** ■

Since the Markov chain is positive recurrent, it has a unique stationary distribution. Let us associate each VOQ of a switch with a non-negative weight $w_{ij}(n)$ (i.e. $w_{ij}(n) = log(Q_{ij}(n))$) at time n. We require that the weight function satisfies the condition that when $Q_{ij}(n) = 0$, $w_{ij}(n) = -\infty$. Define the probability $p_{ij} = \frac{e^{w_{ij}(n)}}{e^{w_{ij}(n)}+1}$, and $p_{ij} = 0$ when $Q_{ij}(n) = 0$. We have the following result.

Lemma 5: The Markov chain of the system has the following product-form stationary distribution:

$$\pi(\mathbf{X}) = \frac{1}{\mathcal{Z}} \prod_{(i,j)\in\mathbf{X}} \frac{p_{ij}}{\overline{p}_{ij}} = \frac{1}{\mathcal{Z}} \prod_{(i,j)\in\mathbf{X}} e^{w_{ij}(n)}, \tag{6}$$

where

$$\mathcal{Z} = \sum_{\mathbf{X}\in\mathcal{X}} \prod_{(i,j)\in\mathbf{X}} \frac{p_{ij}}{\overline{p}_{ij}} = \sum_{\mathbf{X}\in\mathcal{X}} \prod_{(i,j)\in\mathbf{X}} e^{w_{ij}(n)}. \tag{7}$$

Proof: If a state $\mathbf{X}$ can make a transition to $\mathbf{X}'$, we can check that the distribution in Eq. (6) satisfies the detailed balance equation:

$$\pi(\mathbf{X})p(\mathbf{X}, \mathbf{X}') = \pi(\mathbf{X}')p(\mathbf{X}', \mathbf{X}), \tag{8}$$

hence the Markov chain is reversible and Eq. (6) is the stationary distribution (see [13], Theorem 1.2). ■

V. System Stability

One of the most popular algorithms which has been proved stable is the *Maximum Weight Matching (MWM)* algorithm. The **MWM** algorithm selects a feasible schedule with the maximum weight:

$$\mathbf{S}^*(n) = \arg\max_{\mathbf{S}\in\mathcal{S}} \sum_{(i,j)\in\mathbf{S}} w_{ij}(n). \tag{9}$$

The stability result has been shown both for bufferless crossbar switches and buffered crossbar switches [8]. Following the DISQUO algorithm we presented in Sec. III, if $X_{ij}(n) = 1$ and $Q_{ij}(n) > 0$, then $B_{ij}(n) = 1$ $\big($if $B_{ij}(n^-) = 0$, since $Q_{ij}(n) > 0$, input i will send a packet to CB_{ij} at the beginning of n $\big)$, and thus $S^O_{ij}(n) = 1$. A DISQUO schedule is a feasible schedule in bufferless crossbar switches, and if $X_{ij}(n) = 1$ and $Q_{ij}(n) > 0$, one packet is transmitted from input i to output j. Therefore, we can define the weight of a DISQUO schedule as:

$$W(\mathbf{X}) = \sum_i \sum_j X_{ij}(n) w_{ij}(n). \tag{10}$$

For **MWM**, the result below has been established in [14].

Lemma 6: For a scheduling algorithm, if given any ϵ and δ such that $0 \leq \epsilon$, $\delta < 1$, there exists a $B > 0$ such that the scheduling algorithm satisfies the condition that in any time slot n, with a probability greater than $1 - \delta$, the scheduling algorithm can choose a feasible schedule $\mathbf{S}$ which satisfies the following condition:

$$\sum_{(i,j)\in\mathbf{S}(n)} w_{ij}(n) \geq (1-\epsilon) \sum_{(k,l)\in\mathbf{S}^*(n)} w_{kl}(n), \tag{11}$$

whenever $||\mathbf{Q}(n)|| \geq B$, where $\mathbf{Q}(n) = (Q_{ij}(n))$ and $||\mathbf{Q}(n)|| = \big(\sum_{i,j} Q^2_{ij}(n)\big)^{1/2}$. Then the scheduling algorithm can stabilize the system.

Since we already derived the stationary distribution of the Markov chain, we will prove the system stability using Lemma 6. Before the proof of Theorem 1, we first have to prove the lemmas below.

Lemma 7: Suppose that $T(\cdot)$ is a function defined on a set $\mathcal{X}$. For any probability distribution μ on $\mathcal{X}$, define the function:

$$F(\mu, T(\mathbf{X})) = E_\mu[T(\mathbf{X})] + H(\mu), \tag{12}$$

where $H(\mu)$ is the entropy function: $-\sum_{\mathbf{X}\in\mathcal{X}} \mu(\mathbf{X}) \log \mu(\mathbf{X})$. Then $F(\cdot)$ is uniquely maximized by the distribution:

$$\mu^*(\mathbf{X}) = \frac{1}{Z} \exp(T(\mathbf{X})), \tag{13}$$

where $Z = \sum_{\mathbf{X}\in\mathcal{X}} \exp(T(\mathbf{X}))$.

Proof: For any probability distribution μ, we have:

$$\begin{aligned}
& F(\mu, T(\mathbf{X})) \\
= \; & E_\mu[T(\mathbf{X})] + H(\mu) \\
= \; & \sum_{\mathbf{X}\in\mathcal{X}} \mu(\mathbf{X}) T(\mathbf{X}) - \sum_{\mathbf{X}\in\mathcal{X}} \mu(\mathbf{X}) \log \mu(\mathbf{X}) \\
= \; & \sum_{\mathbf{X}\in\mathcal{X}} \mu(\mathbf{X}) (\log \mu^*(\mathbf{X}) + \log Z) - \sum_{\mathbf{X}\in\mathcal{X}} \mu(\mathbf{X}) \log \mu(\mathbf{X}) \\
= \; & \sum_{\mathbf{X}\in\mathcal{X}} \mu(\mathbf{X}) \log Z + \sum_{\mathbf{X}\in\mathcal{X}} \mu(\mathbf{X}) \log \frac{\mu^*(\mathbf{X})}{\mu(\mathbf{X})} \\
\leq \; & \log Z \sum_{\mathbf{X}\in\mathcal{X}} \mu(\mathbf{X}) + \log\Big(\sum_{\mathbf{X}\in\mathcal{X}} \mu(\mathbf{X}) \frac{\mu^*(\mathbf{X})}{\mu(\mathbf{X})} \Big) \\
= \; & \log Z,
\end{aligned} \tag{14}$$

with equality holding only when $\mu = \mu^*$. **QED** ■

Note that when $T(\mathbf{X}) = 0$, the uniform distribution maximizes $F(\mu, 0)$, and we have:

$$F(\mu, 0) = H(\mu) \leq \log Z = \log |\mathcal{X}| \tag{15}$$

where $|\mathcal{X}|$ is the size of $\mathcal{X}$.

Lemma 8: Let $W(\cdot)$ be the weight function and $W^*(\mathbf{X})$ the maximum weight. Define the set:

$$\mathcal{K} = \{\mathbf{X} \in \mathcal{X} : W(\mathbf{X}) \leq (1-\epsilon) W^*(\mathbf{X})\}. \tag{16}$$

Then, we have:

$$\pi(\mathcal{K}) \leq \frac{\log |\mathcal{X}|}{\epsilon W^*(\mathbf{X})} \tag{17}$$

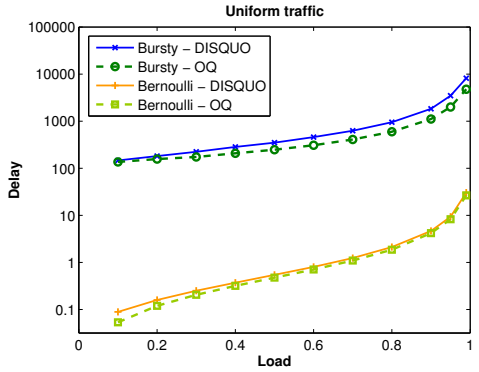

Fig. 3: Switch size N=32, uniform traffic

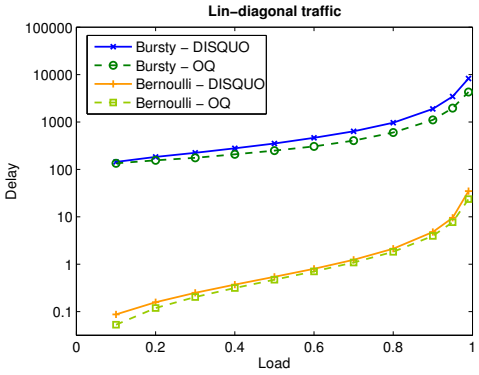

Fig. 4: Switch size N=32, lin-diagonal traffic

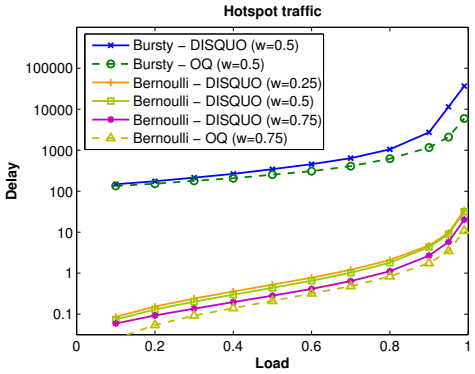

Fig. 5: Switch size N=32, hot-spot traffic

Proof: As shown in Eq. (6), for a schedule $\mathbf{X} \in \mathcal{X}$, its stationary distribution is: $\pi(\mathbf{X}) = \frac{1}{Z}\prod_{(i,j)\in\mathbf{X}} e^{(w_{ij}(n))} = \frac{1}{Z}e^{W(\mathbf{X})}$. According to Lemma 7, π maximizes $F(\mu, W(\mathbf{X}))$.

Let $\mathbf{X}^*$ be the schedule which can give the maximum weight. Let π' be the distribution that assigns all probability on $\mathbf{X}^*$ such that:

$$\pi'(\mathbf{X}) = \begin{cases} 1 & \text{if } \mathbf{X} = \mathbf{X}^* \\ 0 & \text{otherwise} \end{cases}$$

Then we have:

$$\begin{aligned} F(\pi', W(\mathbf{X})) &= E_{\pi'}[W(\mathbf{X})] + H(\pi') \\ &= W^*(\mathbf{X}) + H(\pi') \\ &\leq F(\pi, W(\mathbf{X})) = E_{\pi}[W(\mathbf{X})] + H(\pi) \\ &\leq W^*(\mathbf{X})(1 - \epsilon\pi(\mathcal{K})) + H(\pi) \end{aligned} \quad (18)$$

Last step in Eq. (18) is using Eq. (17). So,

$$\begin{aligned} W^*(\mathbf{X}) + H(\pi') &\leq W^*(\mathbf{X})(1 - \epsilon\pi(\mathcal{K})) + H(\pi) \\ \epsilon\pi(\mathcal{K})W^*(\mathbf{X}) &\leq H(\pi) - H(\pi') \leq H(\pi) \leq \log|\mathcal{X}| \\ \pi(\mathcal{K}) &\leq \frac{\log|\mathcal{X}|}{\epsilon W^*(\mathbf{X})} \end{aligned} \quad (19)$$

■

Theorem 1: DISQUO can stabilize the system if the input traffic is admissible.

Proof: For any $\delta > 0$, we have $\pi(\mathcal{K}) < \delta$, if the maximum weight satisfies the condition:

$$W^*(\mathbf{X}) > \frac{N^2 \log 2}{\epsilon\delta} > \frac{\log|\mathcal{X}|}{\epsilon\delta}. \quad (20)$$

So, for any $\epsilon, \delta > 0$, there exists a $B > 0$ such that whenever $||\mathbf{Q}(n)|| > B$, Eq. (20) holds and then $\pi(\mathcal{K}) < \delta$. Hence the scheduling algorithm can stabilize the system according to Lemma 6. ■

VI. Simulations

In this section, we ran simulations for different scenarios to determine the performance of DISQUO. We also study the delay performance of the scheduling algorithm under different traffic patterns, including uniform and non-uniform traffic with Bernoulli and bursty arrivals. For bursty traffic, the burst length is distributed over $[1, 1000]$, following the truncated Pareto distribution:

$$P(l) = \frac{c}{l^\alpha}, \ l = 1, 2, \ldots, 1000, \quad (21)$$

where l is the burst length, α is the Pareto parameter and c is the normalization constant. In the simulations, $\alpha = 1.7$, for which the average burst length is about 11.6. All inputs are equally loaded and we measure the packet delay.

A. Uniform Traffic

For uniform traffic, a new cell is destined with equal probability to all output ports. Let λ represent the traffic load, the arrival rate between input i and output j is $\lambda_{ij} = \frac{\lambda}{N}$. The delay performance of DISQUO under uniform Bernoulli and bursty traffic is shown in Fig. 3. We can see that the packet delay of DISQUO is very close to the output-queued switch (OQ). It has been shown that under uniform traffic, even an algorithm as simple as RR-RR can have a delay performance close to an output-queued switch [6]. However, the RR-RR algorithm cannot achieve 100% throughput when the traffic is non-uniform. Therefore, we will study the performance of DISQUO under non-uniform traffic next.

B. Non-uniform Traffic

We ran the simulations for the following traffic patterns:

- Lin-diagonal: Arrival rates at the same input differ linearly, i.e, $\lambda_{i(i+j \pmod N)} - \lambda_{i(i+j+1 \pmod N)} = 2\lambda/N(N+1)$.

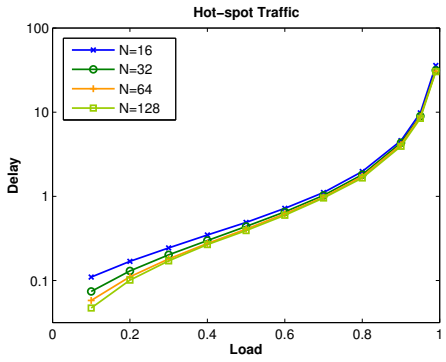

Fig. 6: Impact of switch size, hot-spot traffic, $\omega = 0.5$

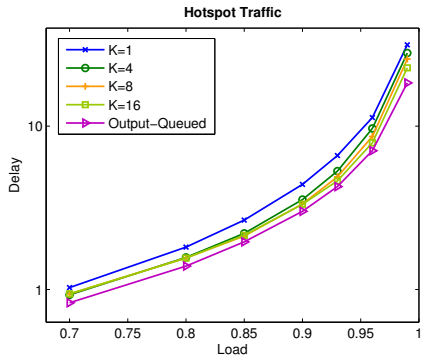

Fig. 7: Impact of buffer size, hot-spot traffic, $\omega = 0.5$, N=32

- Hot-spot: For input port i, $\lambda_{ii} = \omega\lambda$ and $\lambda_{ij} = (1 - \omega)\lambda/(N - 1)$, for $i \neq j$. We can get different traffic patterns by varying the hot-spot factor ω.

The delay performance for lin-diagonal and hot-spot traffic are shown in Fig. 4 and 5, respectively. We can see that under Bernoulli traffic, the delay performance of DISQUO is still very close to the output-queued switch. Packets have low delay even when the load is as high as 0.99. Note that the RR-RR algorithm can have a throughput of approximately only 85% [6] under hotspot traffic. When the traffic arrival is bursty, DISQUO has a delay a little larger than the output-queued switch.

C. Impact of Switch Size

In this section, we will study the impact of switch size on the delay performance. Generally, for input-queued switches, the average delay increases linearly with the switch size. For output-queued switches, delay is almost independent of the size. Fig. 6 shows the delay performance of switches with different sizes under hot-spot traffic, for which ω is 0.5. We can see that, for Bernoulli traffic, the delay is almost the same for different switch sizes. As the size increases, the delay even decreases slightly. This shows that DISQUO is feasible for large scale packet switches.

D. Impact of Buffer Size

If the buffer at each crosspoint increases to infinity, the buffered crossbar switch is then equivalent to an output-queued switch. So as we increase the buffer size, the average delay will decrease and slowly converge to the delay of an output-queued switch. As we already showed in previous simulation results, the delay performance of the new algorithm with buffer size 1 is already very close to that of an output-queued switch. Therefore, by increasing the buffer size, we can only get a very marginal improvement on the delay performance. DISQUO can be easily modified for values of $K > 1$. Due to space considerations, we will not define DISQUO with $K > 1$ here. Fig. 7 shows the delay performance of DISQUO with different buffer sizes, under hot-spot traffic. We can see that the improvement is very small. Therefore, we only need to implement a one-cell buffer at each crosspoint and still provide good delay performance. This is crucial since current technology limits the size of crosspoint buffers to a small number.

VII. Conclusion

In this paper, we propose a distributed scheduling algorithm (DISQUO) for buffered crossbar switch. We prove that it can achieve 100% throughput under any admissible arrival traffic with only a one-packet buffer at each crosspoint. Our simulation results show that it can provide very good delay performance under different traffic arrivals. The simulation results also show that, by using DISQUO, packet delay is very weakly dependent on the switch size, which means that DISQUO can scale with the number of switch ports.

References

[1] J. G. Dai and B. Prabhakar, "The Throughput of Data Switches with and without Speedup," in *Proc. of IEEE INFOCOM*, (Tel Aviv, Israel), March 2000.

[2] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *IEEE Transactions on Communications*, vol. 47, pp. 1260–1267, August 1999.

[3] N. Mckeown, "The iSLIP Scheduling Algorithm for Input-Queued Switches," *IEEE/ACM Transactions on Networking*, vol. 7, pp. 188–201, April 1999.

[4] Y. Li, S. Panwar, and H. J. Chao, "On the Performance of a Dual Round-Robin Switch," in *Proc. of IEEE INFOCOM*, April 2001.

[5] Y. Li, S. Panwar, and H. J. Chao, "Exhaustive Service Matching Algorithms for Input Queued Switches," in *Proc. of IEEE HPSR*, (Phoenix, AZ), April 2004.

[6] R. Rojas-Cessa, E. Oki, and H. J. Chao, "On the Combined Input-Crosspoint Buffered Packet Switch with Round-Robin Arbitration," *IEEE Transactions on Communications*, vol. 53, pp. 1945–1951, November 2005.

[7] T. Javidi, R. Magill, and T. Hrabik, "A High Throughput Scheduling Algorithm for a Buffered Crossbar Switch Fabric," in *Proc. of IEEE ICC*, (Helsinki, Finland), June 2001.

[8] Y. Shen, S. S. Panwar, and H. J. Chao, "Providing 100% Throughput in a Buffered Crossbar Switch," in *Proc. of IEEE HPSR*, (Brooklyn, New York), May-June 2007.

[9] S. Rajagopalan and D. Shah, "Aloha That Works," *submitted*, Nov. 2008.

[10] L. Jiang and J. Walrand, "A Distributed Algorithm for Optimal Throughput and Fairness in Wireless Networks with a General Interference Model," *submitted*, June 2008.

[11] J. Ni and R. Srikant, "Distributed CSMA/CA Algorithms for Achieving Maximum Throughput in Wireless Networks," *submitted*, March 2009.

[12] P. Giaccone, B. Prabhakar, and D. Shah, "Toward Simple, High Performance Schedulers for High-Aggregate Bandwidth Switches," in *Proc. of IEEE INFOCOM*, (New York), 2002.

[13] F. Kelly, *Reversibility and Stochastic Networks*. Wiley, 1979.

[14] A. Eryilmaz, R. Srikant, and J. R. Perkins, "Stable Scheduling Policies for Fading Wireless Channels," *IEEE/ACM Transactions on Networking*, vol. 13, pp. 411–424, April 2005.